Worksheet 1

Introduction VTK

Before we start ...

Here are some valuable tips based on the experience from previous iterations of these labs that might facilitate the tasks in the labs.

1. We will discuss Python, software development and object-oriented programming in these sessions. Make sure that your fundamentals are good, and if you need help understanding the basics (running code, assigning variables, working with classes/objects, etc), try to acquire those skills on your own. Of course, the lab assistant(s) are happy to help with all questions, but we can usually not fix everything in the lab sessions.

2. Structure your projects/directories and files properly and keep your code clean and understandable. For example, try not to name files/variables `test1bbbb.py` but rather give them descriptive names.

3. Don't hard-code filenames and some configurations into the Python files but instead read them from configuration files or use command line arguments.

4. Try not to copy-paste code fragments from the instructions/the user guide but rather type them in by hand. While this might seem cumbersome, it enables a far better understanding of what is happening.

5. Use pycharm as your IDE. If you're fit with another environment like Visual Studio, feel free to use that, but we might not be able to help you as efficiently in case something doesn't work.

6. Sometimes, computers without a dedicated graphics card can run into problems with some parts, especially with the `vtkGPUVolumeRayCastMapper()`. In that case, try to use `vtkVolumeRayCastMapper()` or `vtkFixedPointVolumeRayCastMapper()` instead.

7. If you struggle with a part, something is unclear, or there are mistakes (typos or logical errors) in the instructions, don't hesitate to tell the lab assistant. Happened every year so far :)

8. Play around with different parts of the code (use your values/if you have a specific idea, try to make it happen). It's very hard to break anything that cannot be recovered.

9. For the project: Get used to Git and GitHub early on.

Worksheet 1

Exercise 0 - setup programming environment

The steps below should work for Linux, macOS, or Windows. [1] One more thing, if you decided to use the PyCharm IDE (link) you can install the required packages (vtk, PyQT5, pyvista, pyvistaqt) directly and don't need to do the steps below.

- open a terminal.
- first, check if Python is available: `Python -version`
- if not, install Python for your operating system: link to install instructions
- check if pip works: `pip -version`
- if not, install pip: link to pip install instructions
- install vtk via

```
pip install vtk
```

- install python qt

```
pip install PyQt5
```

- verify that the libraries (vtk and PyQt5) are installed properly by opening a python shell (command: `python`) and typing first `import vtk` and then `import PyQt5`. If the commands run without an error, everything should work. Close the python shell with `ctrl + d`.
- Lastly, we also need two more libraries, pivista and pyvistaqt

```
pip install pyvista pyvistaqt
```

- (optional but recommended if not using PyCharm) install ipython:

```
pip install ipython
```

[1] Note that I recommend working in virtual environments for good practice. We will not cover this in this Lab, however, so it is up to your responsibility.

Worksheet 1

Exercise 1

main_surface.py

run the script with the provided data:

```
ipython main_surface.py ./path/to/your/BraTS19_Testing_069_t1.nii.gz
```

Afterwards, look into the script and try to understand the individual pipeline stages in detail.

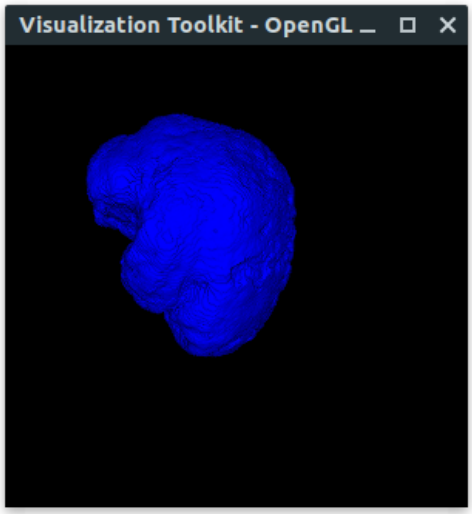

Figure 1: `main_surface.py`: intended outcome

main_volume.py

Now, you have to write a similar script by yourself. Note that the numbers in the comments correspond to the steps in this instruction. Therefore, follow the instructions in the comments in the script `main_volume.py`. Below, you can find hints and additional references for the individual steps.

1 Load the filename from the first argument of the script, i.e., the script should be called like this:

```
ipython main_volume.py ./path/to/your/BraTS19_Testing_069_t1.nii.gz
```

tip: use `sys.argv`. If you need clarification on what command line arguments are and how to use them, I advise you to follow this guide: (link). For Pycharm, look in addition here (link).

Worksheet 1

2 Vtk provides an extra loader for NIFTII data:

```
reader_src = vtk.vtkNIFTIImageReader()
reader_src.SetFileName(filename) # replace filename
```

3 *tip* use

```
vol_map = vtk.vtkGPUVolumeRayCastMapper()
```

as mapper. Remember to set the input connection of the mapper to the reader source.

4-6 In order to understand how the piece-wise linear functions and colour transfer functions work, read VTKUsersGuide (link) Section 7.5 - 7.8.
tip for adapting code from the user guide The hints in the user guide are mostly in plain commands or C++. They can (and have to) be translated into Python. For example, in the VTKUsersGuide on page 145:

```
vtkColorTransferFunction ctfun
    ctfun SetColorSpaceToRGB
    ctfun AddRGBPoint 0 1 0 0
    ctfun AddRGBPoint 127 0 1 0
    ctfun AddRGBPoint 255 0 0 1
```

can be understood in Python as

```
    ctfun = vtk.vtkColorTransferFunction()
    ctfun.SetColorSpaceToRGB()
    ctfun.AddRGBPoint(0, 1, 0, 0)
    ctfun.AddRGBPoint(127, 0, 1, 0)
    ctfun.AddRGBPoint(255, 0, 0, 1)
```

Then, create and connect the `vtk.vtkVolumeProperty` according to the user guide and the values from the code instructions.

7 Create an actor for our volume and connect it to our previously defined mapper and properties:

```
vol_act = vtk.vtkVolume()
volAct.SetMapper(mapper) # connect your own mapper
volAct.SetProperty(property) # connect your own property
```

8 Next, our vtk scene needs a properly positioned camera in the 3D space. Find the right section and implement it following VTKUsersGuide on page 50: *Controlling The View Direction*

9 We must also implement a renderer to view our scene. An example of how to set up a `vtkRenderer` is available, for example, in the VTKUsersGuide on page 43.

10-17 Implement the rest of the script according to the comments. *tip* use `vtk.vtkVolume()` instead of `vtk.vtkActor()` as an actor for the volume.

Note: Take a bit of time here to play with different lookup tables, colour values, etc., until you achieve a representation that you like.

Figure 2: `main_volume.py`: intended outcome

Exercise 2 - multiple sections with PyQT

The goal of this exercise is to turn the script we wrote in exercise 1 into a custom class that we can reuse. Furthermore, we are going to use PyQG in order to build a simple user interface for our application (more on how to build program interfaces in a later lab).

main_qt_multi.py

1 Similar to the previous exercise, run the script `main_qt_multi.py` on the file `BraTS19_Testing_069_t1.nii.gz`.

2-3 note that our current list contains three renderers in one row and tree columns. Create a window that instead uses six renderers in two rows and three columns. *Note:* For now, use only surface renderers.

4 Take a look and try to understand the general content of the file `main_qt_multi.py`. *Note:* For now, you don't have to understand the mechanisms of the parts related to Qt.

5 compare the scripts `main_surface.py` and `renderers/Surface.py`. Afterwards, complete the script `renderers/Volume.py` so that it contains all relevant commands from Exercise 1 to render a volume accordingly. Note that there are small differences in the render window because we provide a QT window now with the frame argument.

6 import `renderers/Volume.py` inside of the `main_qt_multi.py` file:

```
from renderers.Surface import SurfaceRenderer
```

7 Then, append an object of the class to the renderers list:

```
render.append(...)
```

8 Adapt the layout accordingly, e.g. make a window that contains 2 SurfaceRenderers and 2 VolumeRenderers in a (2, 2) layout.

9 Run the script `main_qt_multi.py` and confirm that everything is working correctly, and you can see volume renders along with surface renders.

Exercise 3 - PyVista

There exists a library that wraps a lot of the cumbersome VTK boilerplate code into user-friendly Python functions; - PyVista. It provides a vast number of useful functionalities and examples (link). While PyVista makes many things more accessible, it is also less flexible than directly working with vtk code. It is therefore important to understand both libraries for this course.

1. Let's start with PyVista by running and inspecting the files `main_pyvista_surface.py` and `main_pyvista_volume.py`. Note how fewer lines of code lead to arguably better-looking results than we were able to achieve by using vtk directly.

2. Look at the content of `renderers/SurfacePV.py`. Note that we have to use a special `QtInteractor` from `pyvistaqt` in order to connect PyVista with our QT application. Then, go back to `main_qt_multi.py`, import `SurfaceRendererPV` from `SurfacePV.py`, and add it to the layout as we did in Exercise 2.

3. Lastly, complete the code in `renderers/VolumePV` by looking at `main_pyvista_volume.py` and `renderers/SurfacePV`. Afterwards, the application should be able to add all four different types of renderers, e.g. in a two by two layout as displayed in Figure 3. Hint: With PyVista, we can directly add the output of the reader source to the plotter by calling

   ```
   self.plotter.add_volume(reader_source.GetOutput())
   ```

4. (optional) think about and research more ways to visualize the data, e.g. slice-rendering or interactive vtk widgets. If you find something you like, e.g. an online example or another visualization from PyVista, try to a) write a main script that uses the provided data or b) create your own renderer class so that you can use it in the window from Exercise 2.

Learning outcome of this Lab

1. Understand the vtk pipeline and all the modules like mappers, renderers, etc.

2. Be able to implement and configure surface and volume rendering in vtk and PyVista.

3. Understand how we used Python modules, classes, and imports to structure the code.

4. Get a first idea of PyQT to build application user interfaces.

Worksheet 1

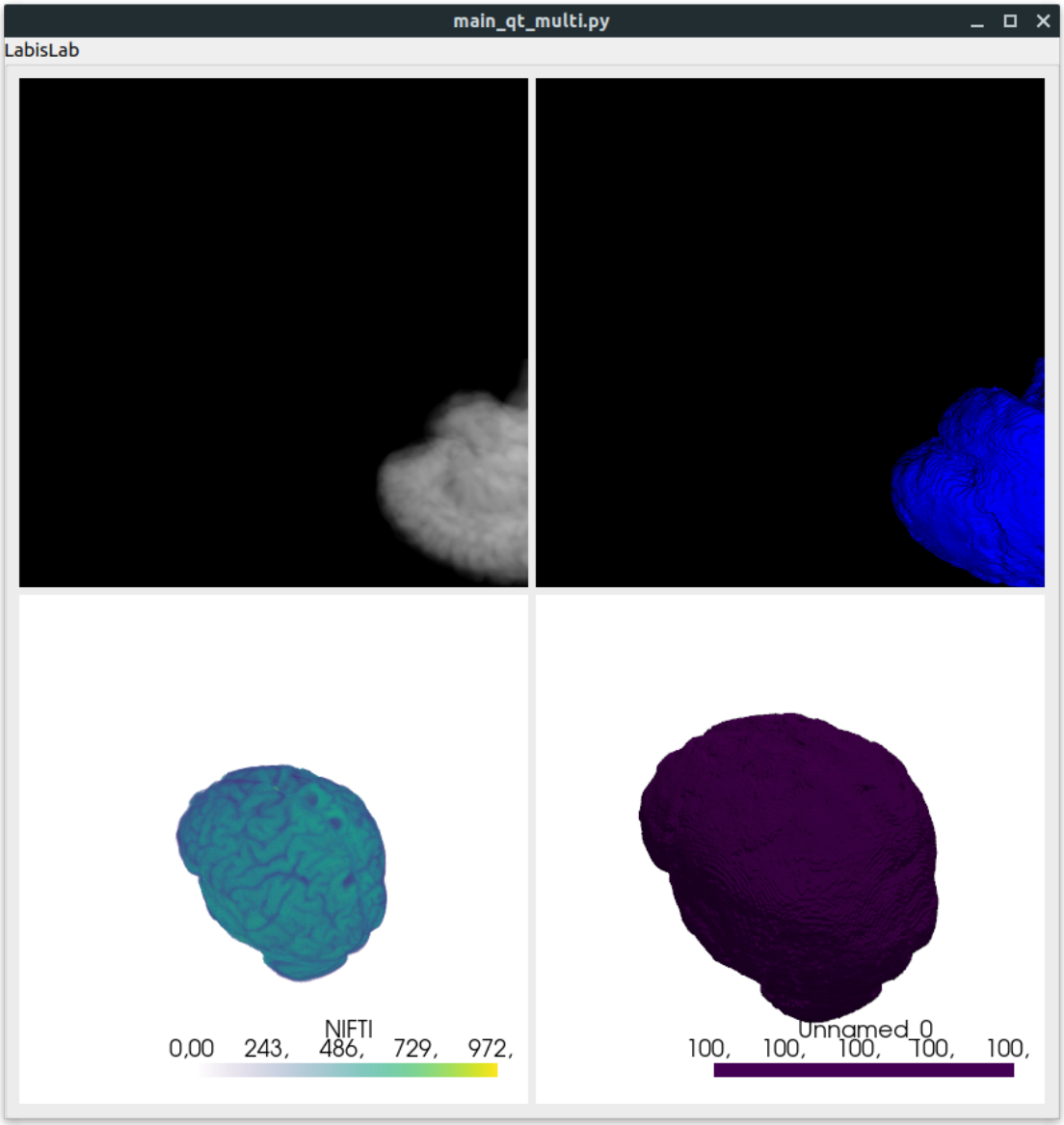

Figure 3: `main_qt_multi.py: intended outcome`